SKRIPSI

ANALISIS KINERJA WEB SERVICES PADA MICROSERVICES BERBASIS GOLANG (STUDI KASUS DATA SISTER WEB SERVICES PT 1.0.0)

Disusun dan diajukan oleh:

REZA ARISANDY SAFRUDDIN
D121 18 1 016

PROGRAM STUDI SARJANA TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS HASANUDDIN
GOWA
2024

LEMBAR PENGESAHAN SKRIPSI

ANALISIS KINERJA WEB SERVICES PADA MICROSERVICES BERBASIS GOLANG (STUDI KASUS DATA SISTER WEB SERVICES PT 1.0.0)

Disusun dan diajukan oleh

Reza Arisandy Safruddin
D121181016

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian Studi Program Sarjana Program Studi Departemen Teknik Informatika
Fakultas Teknik Universitas Hasanuddin
Pada tanggal 2 Februari 2024
dan dinyatakan telah memenuhi syarat kelulusan

Menyetujui,

Pembimbing Utama,	Pembimbing Pendamping,
Dr. Eng. Ir. Muhammad Niswar, S.T., M.InfoTech	Anugrayani Bustamin, S.T., M.T
NIP 197309221999031001	NIP 199012012018074001

Ketua Program Studi,

Prof. Dr. Ir. Indrabayu ST, MT, M.Bus.Sys., IPM, ASEAN. Eng
NIP 197507162002121004

PERNYATAAN KEASLIAN

Yang bertanda tangan dibawah ini ;
Nama : Reza Arisandy Safruddin
NIM : D121181016
Program Studi : Teknik Informatika
Jenjang : S1

Menyatakan dengan ini bahwa karya tulisan saya berjudul

{ Analisis Kinerja Web Services Pada Microservices Berbasis Golang (Studi Kasus Data Sister Web Services Pt 1.0.0) }

Adalah karya tulisan saya sendiri dan bukan merupakan pengambilan alihan tulisan orang lain dan bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri.

Semua informasi yang ditulis dalam skripsi yang berasal dari penulis lain telah diberi penghargaan, yakni dengan mengutip sumber dan tahun penerbitannya. Oleh karena itu semua tulisan dalam skripsi ini sepenuhnya menjadi tanggung jawab penulis. Apabila ada pihak manapun yang merasa ada kesamaan judul dan atau hasil temuan dalam skripsi ini, maka penulis siap untuk diklarifikasi dan mempertanggungjawabkan segala resiko.

Segala data dan informasi yang diperoleh selama proses pembuatan skripsi, yang akan dipublikasi oleh Penulis di masa depan harus mendapat persetujuan dari Dosen Pembimbing.

Apabila dikemudian hari terbukti atau dapat dibuktikan bahwa sebagian atau keseluruhan isi skripsi ini hasil karya orang lain, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Gowa, 2 Februari 2024

Menyatakan

Reza Arisandy Safruddin

ABSTRAK

REZA ARISANDY SAFRUDDIN. *Analisis Kinerja Web Services Pada Microservices Berbasis Golang (Studi Kasus Data Sister Web Services Pt 1.0.0)* (dibimbing oleh Dr. Eng. Ir. Muhammad Niswar, S.T., M.InfoTech dan Anugrayani Bustamin, S.T., M.T)

Pada era 4.0, penggunaan *microservice* mengalami peningkatan signifikan dibandingkan dengan era 3.0. Arsitektur *microservice* yang berkembang dianggap lebih baik daripada pendekatan monolitik. Era ini ditandai oleh peningkatan pemahaman dan penerapan praktik-praktik modern dalam pengembangan perangkat lunak, yang bertujuan untuk mengatasi tantangan yang timbul dalam lingkungan teknologi yang semakin kompleks. Pengembangan *microservice* yang efektif menjadi krusial dalam membangun sistem yang efisien dan dapat diskalakan. Dalam upaya ini, terdapat tiga metode komunikasi umum yang digunakan: Representational State Transfer (REST), gRPC, dan GraphQL. Penelitian ini melakukan pengujian terhadap ketiga metode berdasarkan *Throughput*, *Response Time*, dan *CPU Utilization*. Studi kasus yang diambil adalah Data sister Universitas Hasanuddin dengan 2 skenario, dan tiap skenario berdasarkan jumlah request (100 – 500 Request). Hasil penelitian menunjukkan bahwa gRPC memiliki *response time* yang lebih cepat, diikuti oleh metode REST dan GraphQL. Begitu pula dalam pengujian berdasarkan *Throughput*, gRPC memiliki *Throughput* yang lebih tinggi daripada REST dan GraphQL. Namun, pada pengujian penggunaan *CPU Utilization*, GraphQL memiliki *CPU Utilization* yang lebih tinggi dibandingkan dengan metode gRPC dan REST.

Kata kunci: *Microservices*, API, REST API, gRPC, GraphQL

ABSTRACT

REZA ARISANDY SAFRUDDIN. *Performance Analysis of Web Services in Golang-Based Microservices (Case Study: Data Sister Web Services Pt 1.0.0)* (supervised by Dr. Eng. Ir. Muhammad Niswar, S.T., M.InfoTech and Anugrayani Bustamin, S.T., M.T)

In the era 4.0, the use of microservices has seen a significant increase compared to the era 3.0. The evolving microservices architecture is considered superior to the monolithic approach. This era is characterized by an improved understanding and implementation of modern practices in software development, aimed at addressing challenges arising in an increasingly complex technological environment. Effective development of microservices becomes crucial in building efficient and scalable systems. In this endeavor, three common communication methods are employed: Representational State Transfer (REST), gRPC, and GraphQL. This research conducts testing on these three methods based on Throughput, Response Time, and CPU Utilization. The case study involves the sister Data of Hasanuddin University with two scenarios, each scenario based on the number of requests (100 - 500 Requests). The research results indicate that gRPC has a faster response time, followed by the REST and GraphQL methods. Similarly, in the Throughput testing, gRPC exhibits higher Throughput compared to REST and GraphQL. However, in the CPU Utilization testing, GraphQL shows higher CPU Utilization than the gRPC and REST methods.

Keywords: Microservices, API, REST API, gRPC, GraphQL

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR SINGKATAN DAN ARTI SIMBOL

Lambang/Singkatan	Arti dan Keterangan
REST	*Representational State Transfer*
API	*Application Programming Interface*
JWT	*Json Web Tokens*
IDL	*Interface Definition Language*
HTTP	*Hypertext Transfer Protocol*
Protobuf	Protokol Buffer
JSON	*JavaScript Object Notation*
RPC	*Remote Procedure Call*
GUI	*Graphical User Interface*
CLI	*Command Line Interface*
TCP	*Transmission Control Protocol*
IP	*Internet Protocol*

DAFTAR LAMPIRAN

KATA PENGANTAR

Assalamu'alaikum Warahmatullahi Wabarakatuh.

Segala puji dan syukur kami panjatkan ke hadirat Allah S.W.T Tuhan Yang Maha Esa yang dengan limpahan rahmat dan hidayah-Nya sehingga tugas akhir dengan judul " Analisis Kinerja Web Services Pada Microservices Berbasis Golang (Studi Kasus Data Sister Web Services Pt 1.0.0) " ini dapat diselesaikan sebagai salah satu syarat dalam menyelesaikan jenjang Strata-1 pada Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin.

Dalam penyusunan penelitian ini disajikan hasil penelitian terkait judul yang telah diangkat dan telah melalui proses pencarian dari berbagai sumber baik jurnal penelitian, prosiding pada seminar-seminar nasional/internasional, buku maupun dari situs-situs di internet.

Penulis menyadari bahwa tanpa bantuan dan bimbingan dari berbagai pihak, mulai dari masa perkuliahan sampai dengan masa penyusunan tugas akhir, sangatlah sulit untuk menyelesaikan tugas akhir ini. Oleh karena itu, pada kesempatan ini penulis menyampaikan ucapan terima kasih sedalam-dalamnya kepada :

1) Tuhan Yang Maha Esa atas semua berkat, karunia serta pertolongan-Nya yang tiada batas, yang telah diberikan kepada penulis disetiap langkah dalam pembuatan program hingga penulisan laporan skripsi ini.
2) Kedua orang tua penulis serta saudara penulis, serta keluarga yang senantiasa memberikan kekuatan, motivasi, bimbingan moral, materi, kepercayaan dan kasih sayang yang tidak terbatas kepada penulis.
3) Bapak Dr. Eng. Ir. Muhammad Niswar, S.T., M.InfoTech. selaku pembimbing I yang telah memberikan banyak bimbingan, motivasi dan masukan yang bermanfaat kepada penulis.
4) Ibu Anugrayani Bustamin, S.T., M.T., selaku pembimbing II yang telah memberikan banyak bimbingan, motivasi dan masukan yang bermanfaat kepada penulis.

5) Bapak Prof. Dr. Ir. Indrabayu S.T., M.T., M.Bus.Sys., IPM, ASEAN. Eng., selaku Ketua Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin atas bantuan dan bimbingannya selama masa perkuliahan penulis.
6) Bapak A. Ais Prayogi Alimuddin, S.T., M.Eng., selaku dosen pembimbing akademik yang telah memberikan bimbingan selama masa perkuliahan penulis.
7) Bapak dan ibu dosen Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin atas bimbingan, arahan dan didikannya selama masa perkuliahan.
8) Bapak Robert, Bapak Zainuddin, dan Ibu Yuanita serta segenap staff Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah membantu kelancaran penyelesaian tugas akhir penulis.
9) Teman-teman *Synchoronous* 2018 yang telah memberikan nasihat, bantuan dan semangat selama proses penyelesaian tugas akhir ini.
10) Semua Orang yang telah membantu dan memberikan dukungan kepada penulis namun tidak sempat disebutkan.

Akhir kata, penulis berharap semoga Tuhan Yang Maha Esa berkenan membalas segala kebaikan dari semua pihak yang telah banyak membantu. Semoga tugas akhir ini dapat memberikan manfaat bagi pengembangan ilmu selanjutnya. Aamiin.

Wassalamu’alaikum Warahmatullahi Wabarakatuh

Gowa, 2 Februari 2024

Penulis.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Microservices merupakan pendekatan arsitektur untuk membangun aplikasi besar sebagai kumpulan beberapa layanan independen dan terisolasi yang saling bekerja sama. Setiap layanan memiliki tugas dan tanggung jawab tertentu dan dapat dikembangkan, di-*deploy*, dan di *maintenance* secara independen. Ini memungkinkan tim pengembangan untuk bekerja secara efisien dan mengurangi wkatu untuk memperbaiki masalah dalam aplikasi. *Microservices* juga memungkinkan aplikasi untuk skalabilan dan evolusi secara mudah karena setiap layanan dapat ditingkatkan atau diterapkan perubahan tanpa mempengaruhi layanan lain. *Microservices* memerlukan integrasi yang efektif dan komunikasi yang baik antar layanan untuk bekerja dengan efisien.

Microservices berkembang dari pendekatan monolitik dalam pengembangan aplikasi. Dalam pendekatan monolitik, seluruh aplikasi dibuat dalam satu entitas besar dan kompleks yang sulit dikelola dan dikembangkan. Ini menyebabkan masalah seperti waktu pengembangan yang lama, kesulitan dalam skalabilan dan pembaruan aplikasi, dan ketergantungan antar bagian aplikasi yang tinggi.

Microservices memungkinkan pengembangan aplikasi secara modular dan mengurangi ketergantungan antar bagian, sehingga mempermudah pembaruan dan perbaikan aplikasi. Oleh karena itu, penggunaan *microservices* dapat membantu memastikan kesuksesan aplikasi dan memberikan keuntungan bisnis dalam jangka panjang.

Microservices juga didukung oleh perkembangan teknologi dan cloud *computing* yang membuat implementasi dan manajemen layanan lebih mudah dan efisien. Dalam beberapa tahun terakhir, *microservices* menjadi pendekatan populer dalam pengembangan aplikasi dan digunakan dalam berbagai jenis aplikasi termasuk e-*commerce*, finansial dan aplikasi seluler.

Pada saat arsitektur *microservices* semakin populer, standar bagaimana layanan tersebut berkomunikasi juga berkembang. Pendekatan yang umum adalah dengan menggunakan REST atau *Representational State Transfer* melalui JSON sebagai IDL atau *Interface Definition Language* (Gonzales, 2019). Istilah REST di perkenalkan pada tahun 2000 dalam disertasi doktoral Roy Felding salah satu

penulis utama spesifikasi *Hypertext Transfer Protocol* (HTTP). REST adalah salah satu gaya arsitektur untuk mengembangkan sistem yang memungkinkan aplikasi client dapat mengakses data dan fungsional yang telah di definisikan oleh suatu sistem (Masse, 2011).

Selain REST, gRPC adalah alternatif lain yang dapat digunakan sebagai metode komunikasi dalam sistem *microservices*. gRPC adalah kerangka kerja RPC (*Remote Procedure Call*) yang bersifat *Open Source* yang dikembangkan oleh insinyur perangkat lunak di Google dan resmi dirilis pada agustus 2016 (Rebrošová 2021). gRPC mendukung berbagai jenis data streaming dengan performa tinggi dan pengiriman data yang cepat karena didukung oleh penggunakan protokol buffer (protobuff) dan HTTP/2 untuk trasnfer data (Stefanic, 2021). Protokol buffer merupakan sebuah teknologi *open source* untuk melakukan serialisasi data. Berbeda dengan JSON, protokol buffer memiliki performa lebih baik karena melakukan serialisasi data menjadi format binary (Berge dkk, 2019). Protokol buffer juga mempertegas kepada pengembang bagaimana komunikasi antar gRPC akan dibangun (Mateus dkk, 2020).

Adapun REST yang *response* dan *request* nya mirip dengan gRPC dalam desain API yakni mengembalikkan data ekstra dari server, namun desain API yang lebih spesifik dalam mengambil data yakni GraphQL. GraphQL merupakan sebuah spesifikasi tingkat aplikasi yang memungkinkan pengembang untuk meminta data yang mereka butuhkan dari server melalui sebuah API. Dikembangkan oleh Facebook pada tahun 2012, GraphQL menawarkan solusi alternatif untuk menangani data yang kompleks dan untuk mengatasi masalah seperti *over-fetching* atau *under-fetching* data. GraphQL adalah mekanisme untuk melakukan interaksi antara client dan server secara remote dengan mengirimkan query ke sebuah endpoint (“Introduction to GraphQL,”.2015).

Go adalah salah satu bahasa yang umum digunakan untuk mengimplementasikan *microservices*. Ada beberapa keuntungan menggunakan Go, termasuk kemampuan konkurensi yang baik, penyetelan koleksi sampah minimal, kompilasi ke biner asli yang membuatnya performan, pendekatan minimalis dengan sintaks mirip Python yang memudahkan pembelajaran, dan dukungan alat yang kaya serta komunitas open-source yang berkembang(Milind Chabbi, M. Ramanathan, 2022).

Pada Go memiliki fitur *concurrency* sehingga memudahkan dalam hal implementasi REST dan gRPC. Jika dibandingkan dengan Node.js, Go memiliki kinerja lebih baik pada microservices. Dikarenakan Go merupakan Bahasa yang di-*compile* ke Bahasa mesin sedangkan Node.js menggunakan JavaScript yang merupakan Bahasa yang diinterpretasi(Salahuddin, 2022)

Maka dari itu, penulis mengajukan penelitian untuk melakukan analisa perbandingan kinerja antara REST API, gRPC, dan GraphQL pada pertukaran data dalam sebuah sistem *microservice* dengan skenario data bersarang (*nested data*) dan data tidak bersarang (*flat data*). Judul dari penelitian tersebut adalah "Analisis Kinerja Web Services Pada Microservices Berbasis Golang (Studi Kasus Data Sister Web Services Pt 1.0.0)". Pada penelitian ini, penulis melakukan analisis kinerja dengan parameter kinerja yang diukur adalah *Response time*, *Throughput*, dan *CPU Utilization.*

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah, maka rumusan masalah pada penelitian tugas akhir ini adalah :

1. Bagaimana mengimplementasikan *microservices* pada Bahasa Pemrograman Go?
2. Bagaimana perbandingan performa *microservices* yang diimplementasikan menggunakan arsitektur API seperti REST API, gRPC dan GraphQL pada bahasa pemrograman Go?

1.3 Tujuan Penelitian/Perancangan

Tujuan dari penelitian ini adalah :

1. Dapat mengimplementasikan Microservices pada bahasa pemrograman Go.
2. Dapat membandingkan performa microservices yang diimplementasikan menggunakan arsitektur API seperti REST API, gRPC dan GraphQL pada bahasa Go.

1.4 Manfaat Penelitian/Perancangan

Manfaat yang didapat dari penelitian ini adalah :

1. Hasil analisis dapat dijadikan sebagai pertimbangan dalam melakukan pengembangan sistem yang menggunakan microservices

2. Membagikan data hasil analisis kinerja microservices yang menggunakan arsitektur Api yakni REST API, gRPC, dan GraphQL berupa data perbandingan performa.

1.5 Ruang Lingkup/Asumsi perancangan

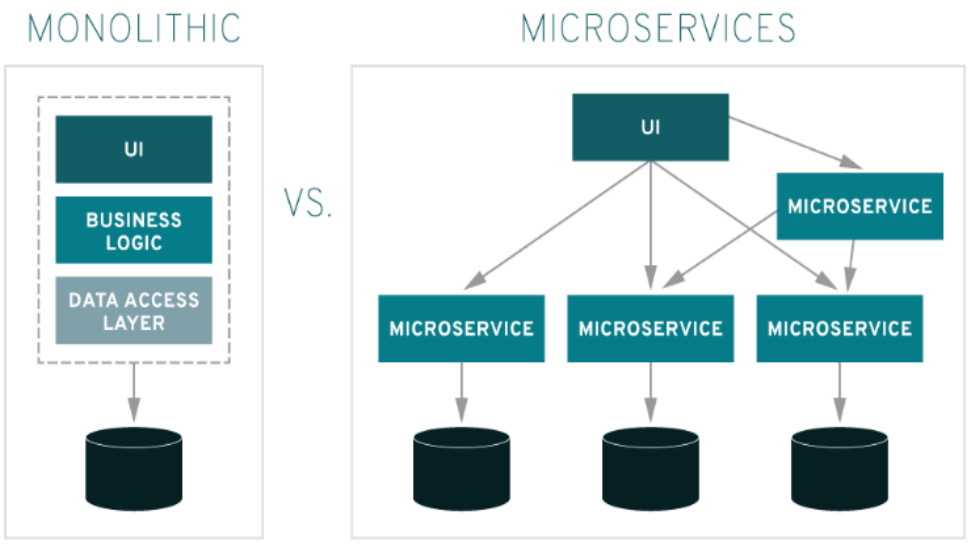
Adapun batasan masalah pada penelitian ini :

1. Studi kasus dari penelitian ini menggunakan data Sister Universitas Hasanuddin versi 1.0.0.
2. Penelitian ini menganalisis performa arsitektur Api (pertukaran data) antara REST API, gRPC dan GraphQL.
3. Pada *authentication* menggunakan JWT (*JSON Web Tokens*) untuk interaksi antar services yang lainnya.
4. Parameter performa terbaik dari tiap kinerja Web Services di ukur berdasarkan *response time, throughput*, dan *CPU Utilization*.
5. Pengukuran kinerja menggunakan Apache JMeter.
6. Services dibuat dengan menggunakan bahasa Go.
7. Database yang digunakan disimpan di redis untuk digunakan pada penelitian

BAB II
TINJAUAN PUSTAKA

2.1 Microservices

Microservices merupakan salah satu pendekatan untuk mengembangkan sebuah aplikasi. Gaya arsitektur yang digunakan pada *microservices* adalah dengan mengembangkan aplikasi perangkat lunak yang kompleks dari kumpulan layanan-layanan kecil. Setiap layanan kecil tersebut bertanggung jawab untuk menyelesaikan tugas tertentu. Layanan tersebut dapat dikembangkan dengan berbagai bahasa pemrograman oleh tim yang terpisah. (Aksakalli, Turgay, Can & Bedir, 2021). Sehingga, arsitektur *microservices* memiliki kelebihhan dibandingkan arsitektur monolitik karena keseluruhan sistem yang dipecah menjadi bagian-bagian kecil yang berjalan independen (shafabakhsh, 2020).



Gambar 2.1 Perbedaan arsitektur microservice dan monolitik (shafabakhsh, 2020)

Pada gambar 2.1 dijelaskan perbedaan arsitektur *microservice* dan monolitik. Pada monolitik model arsitektur sistemnya menempatkan semua fungsi dan fitur dalam satu aplikasi tunggal yang besar. Aplikasi ini biasanya dijalankan di satu server dan diakses melalui satu *endpoint*. Jadi pada satu aplikasi tunggalnya terdapat UI sebagai tampilan, dan logika bisnis suatu sistem yang saling berinteraksi dengan database. Sedangkan pada *microservice* model arsitektur aplikasi yang memisahkan aplikasi menjadi beberapa komponen yang independent yang dapat di-*deploy* dan diubah tanpa mempengaruhi komponen lainnya.

Pengembangan aplikasi menggunakan arsitektur *microservices* tidak memiliki patokan yang baku. Namun, menurut Shafaback ada beberapa karakteristik aplikasi yang harus dimiliki agar dapat dikatakan menggunakan arsitektur *microservices*. Karakter tersebut tidak terbatas pada ukuran setiap services yang relatif kecil, terikat dalam konteks, terdistribusi dan terakhir dapat digunakan secara mandiri. Sehingga shafaback beranggapan bahwa arsitektur *microservices* dapat memberi manfaat seperti kemudahan menggunakan berbagai teknologi seperti bahasa pemrograman atau *framework*, kemudahan untuk melakukan penskalaan, ketersediaan yang tinggi dan meningkatkan keselarasan dalam organisasi (shafabakhsh, 2020).

Bersama kelebihan yang diberikan *microservices*, arsitektur ini juga memiliki beberapa tantangan pada pengaplikasiannya. Sebagaimana yang dijelaskan Richardson (2019) dalam buku "*Microservicces Pattern: with examples in Java"* seperti bagaimana menentukan susunan layanan yang tepat, bagaimana mengembangkan, menguji, menyebarkan hingga bagaimana membangun komunikasi antar services. Pada arsitektur *microservices* sendiri terdapat dua metode komunikasi *synchronous* yang paling umum digunakan yaitu menggunakan REST atau menggunakan gRPC

2.2 REST (*Representational State Transfer*)

REST (Representational State Transfer**)** adalah sebuah arsitektur pengembangan API (*Application Programming Interface*) yang menyediakan komunikasi berbasis *client-server* melalui protokol HTTP (Prayogi, Niswar, Indrabayu & Rija, 2019). REST pertama kali diperkenalkan oleh Roy Fielding pada tahun 2000 sebagai disertasi doktoralnya di University of California. Disertasi tersebut berjudul "*Architectural Styles and the Design of Network-based Software Architecture".* Dalam disertasi tersebut Fielding memperkenalkan REST yaitu sebuah gaya arsitektur untuk sistem hypermedia yang terdistribusi (Doglio, 2018).

REST menggunakan protokol HTTP/1.1 untuk mengirimkan data dari klien ke peladen (Masse, 2012). Dalam sistem yang menggunakan REST, setiap services biasanya memiliki endpoint tertentu agar bisa melakukan interaksi antar service dan bertukar data (Shafabakshs, 2020) . Pada REST terdapat beberapa method yang dapat digunakan, namun yang populer digunakan adalah :

a. GET untuk menerima data yang tersedia

b. POST untuk menambahkan data baru

c. PUT untuk memperbaharui data yang telah ada

d. DELETE untuk menghapus data

Pada REST mendukung beberapa format untuk mempresentasikan data seperti JSON dan XML. Namun, JSON lebih sering digunakan karena lebih muda dipahami. Hal ini karena JSON adalah format data yang berbasis teks (Stefanic, 2021).

Sebagai sebuah gaya arsitektur yang terdistribusi, menurut Doglio (2018) dalam buku "Rest API Development with node.js" REST mampu meningkatkan beberapa hal berikut :

- *Performance*: Gaya komunikasi yang digunakan pada REST ditujukan untuk efisiensi dan sederhana. Sehingga dapat mengoptimalkan kinerja dari sistem yang menggunakanya.
- *Scalability of component interaction*: Gaya komunikasi REST adalah bersifat terdistribusi dan saling terpisah. Sehingga memudahkan untuk melakukan *scalability* dari interaksi antar komponen.
- *Simplicity of interface*: Sebuah antarmuka yang sederhana akan memudahkan sistem menggunakan arsitektur ini.
- *Modifiability of components*: Sistem terdistribusi yang memecah konsentrasi memungkinkan setiap komponen dapat dikembangkan dengan resiko dan biaya yang kecil.
- *Portability*: REST adalah teknologi yang bersifat *language-agnostic*, sehingga teknologi ini dapat digunakan pada bahasa apa saja.
- *Reliability*: REST yang berbasi *stateless* memungkin proses pemulihan dari sebuah sistem yang gagal menjadi lebih mudah.
- *Visibility*: Dengan sistem *stateless*, REST juga memudahkan untuk melakukan proses *monitoring* karena sistem tidak perlu melihat lebih jauh untuk mengetahui keadaan dari sebuah request.

2.3 gRPC

GRPC merupakan salah satu jenis RPC yang sering dugunakan dan terus dikembangkan. RPC (*Remote Procedure Call*) adalah suatu bentuk komunikasi *synchronous* tingkat bahasa untuk melakukan transfer pengendalian antar program

dengan cara memanfaatkan kanal komunikasi terbatas pada sistem terdistribusi (Nelson, 1981). Jadi gRPC adalah teknologi komunikasi antar proses untuk menggabungkan atau memanggil sistem yang terdistribusi semudah memanggil fungsi lokal (Inrasiri & Kuruppu, 2020). GRPC bersifat *open source* yang pada awalnya dikembangkan oleh insinyur perangkat lunak di Google dan resmi dirilis pada Agustus 2016.

Pada gRPC *transfer protocol* yang digunakan adalah HTTP/2.0. Sedangkan format data yang dikirimkan menggunakan protokol buffer (protobuf) (Stefanic, 2021). Protokol buffer adalah sebuah teknologi *open source* untuk melakukan serialisasi data. Berbeda dengan JSON, protokol buffer memiliki performa lebih baik karena melakukan serialisasi data menjadi format *binary* (Berge dkk, 2019). Protokol buffer juga mempertegas kepada pengembang bagaimana komunikasi antar gRPC akan dibangun (Araújo dkk, 2020). Tedapat 4 metode pengiriman data pada gRPC menurut Kasun Indrasiri dan Danesh Kuruppu (2020) antara lain sebagai berikut :

a. Unary RPC

   Unary adalah metode dimana klien yang mengirimkan sebuah permintaan tunggal ke peladen kemudian juga mendapatkan respon yang tunggal. Penggunaan Unary seperti pada pemanggilan fungsi pada umumnya.

b. Client Streaming RPC

   Client Streaming RPC adalah metode dimana klien mengirimkan rangkaian permintaan ke peladen. Selanjutnya peladen akan menunggu semua permintaan sebelum meresponnya secara tunggal.

c. Server Streaming RPC

   Pada metode Server Streaming, klien hanya mengirimkan sebuah permintaan tunggal ke peladen. Namun, peladen akan merespon dengan rangkaian balasan yang berurutan ke klien. Klien kemudian membaca respon tersebut secara berurutan sampai tidak ada lagi respon yang masuk.

d. Bidirectional Streaming RPC

   Metode Bidirectional Streaming memungkinkan klien dan peladen sama-sama mengirimkan pesan. Namun, keduanya tetap dapat beroperasi secara terpisah dan independen sehingga keduanya dapat menerima dan mengirimkan pesan sesuai dengan kebutuhan mereka.

2.4 GraphQL

GraphQL merupakan sebuah bahasa kueri untuk API yang dikembangkan oleh Facebook dan dipergunakan dalam komunikasi antara klien dan server. Kueri meminta data yang dibutuhkan oleh klien, sehingga klien dapat menerima seluruh data yang dibutuhkan dari server dalam satu kali *request*. GraphQL menawarkan solusi alternatif untuk REST API dan memungkinkan pengembang untuk meminta data yang spesifik dalam format yang lebih efisien dan fleksibel. Latar belakang pengembangan GraphQl adalah untuk memenuhi kebutuhan Facebook dalam menangani data yang kompleks dan untuk mengatasi masalah-masalah yang ada pada REST API, seperti over-fetching atau under-fetching data.

Salah satu keuntungan utama GraphQL adalah fleksibilitasnya. Dengan GraphQL, klien dapat meminta banyak sumber data dalam satu permintaan, mengurangi jumlah permintaan yang diperlukan untuk mengambil data yang diinginkan. Selain itu, dengan menggunakan tipe yang didefinisikan secara jelas, klien dapat memvalidasi permintaan kueri mereka sebelum mengirimkannya ke server. Dalah (The GraphQL Foundation, 2020), Beberapa keunggulan yang dimiliki GraphQl, yaitu:

- Kueri dari GraphQL selalu mengembalikan hasil yang dapat diprediksi, membuat aplikasi yang mengimplementasikan GraphQL berjalan cepat dan stabil dikarenakan GraphQL mengontrol data alih-alih server.
- GraphQL API mendapatkan semua data yang dibutuhkan aplikasi dalam satu permintaan (*single request*). Dengan menggunakan GraphQL, aplikasi dapat berjalan dengan cepat bahkan pada koneksi jaringan yang lambat.
- GraphQL API diatur berdasarkan *types* dan *fields*, bukan dengan *endpoints*. GraphQL menggunakan *types* untuk memastikan aplikasi hanya meminta data yang diperlukan dan menghindari kesalahan yang dapat terjadi. Selain itu, aplikasi dapat menggunakan *types* untuk menghindari penulisan *manual parsing code*.
- GraphQL menyediakan beberapa *tools* yang dapat memudahkan interaksi dengan GraphQL API, seperti GraphiQL dan GraphQL Playground. Tools tersebut memudahkan pengguna GraphQL dalam mengetahui data apa saja yang dapat di-*request* dari API tanpa meninggalkan editor, membantu dalam menemukan masalah-masalah potensial sebelum mengirimkan kueri, dan bertanggung jawab dalam meningkatkan kecerdasan kode.

- GraphQL memungkinakn pengembang aplikasi untuk mengembangkan API tanpa memperbaharui API *version*, dimana pengembang dapat menambahkan *fields* dan *types* baru ke GraphQL API tanpa memengaruhi kueri yang sudah ada, sehingga memberikan aplikasi akses yang berkelanjutan ke fitur-fitur baru dan membuat kode server yang lebih bersih dan dapat dengan mudah dikelola.
- GraphQL membuat seluruh API seragam tanpa dibatasi oleh mesin penyimpanan tertentu. Pengembang dapat menyediakan fungsi untuk setiap *fields* pada *type system*, kemudian GraphQL memanggil fungsi-fungsi tersebut dengan konkurensi yang optimal.

Server GraphQL menggunakan *schema* untuk menggambarkan bentuk dari grafik data yang dibuat. Schema mendefinisikan hierarki *type* dan *field* yang diisi dari penyimpanan data *back-end*. Schema juga menentukan secara spesifik kueri dan mutasi apa yang tersedia bagi klien untuk dieksekusi terhadap grafik data.

Dalam mendefinisikan *type*, GraphQL menggunakan bahasa untuk mendefinisikan schema, yang disebut *Schema Definition Language* atau SDL. *Schema Definition Language* (SDL) adalah sintaks yang digunakan dalam GraphQL untuk mendefinisikan schema GraphQL secara deklaratif. SDL memungkinkan pengembang untuk menggambarkan tipe data, bidang, argumen, dan operasi yang tersedia dalam schema GraphQL.

SDL menggunakan sintaks yang mirip dengan JSON untuk mendefinisikan tipe data, termasuk tipe skalar, tipe objek, tipe enumerasi dan tipe yang tidak bernama. Dengan SDL, pengembang dapat mendefinisikan bidang-bidang yang ada dalam tipe objek, tipe data yang dikembalikan oleh bidang tersebut, dan argumen yang dapat diterima oleh bidang saat diminta.

Gambar 2.2 Contoh penggunaan SDL dalam mendefinisikan tipe objek

Pada gambar 2.2 diberikan contoh penggunaan Schema untuk mendefinisikan tipe objek. Schema GraphQL memiliki berbagai macam komponen didalamnya, yaitu sebagai berikut :

1. Types

   Type merupakan unit utama dari GraphQL Schema. *Type* merepresentasikan data dari aplikasi. Sebuah *Type* memiliki *fields* yang merepresentasikan data yang berhubungan dengan setiap objek, yang mana setiap *field* mengembalikan spesifikasi tipe data (dapat berarti Integer atau String, namun dapat juga berupa tipe data buatan atau list tipe).

2. Scalar Types

   Scalar Types (seperti *Int, Float, String, Boolean, ID*) merupakan bagian dari GraphQL yang sangat bermanfaat, namun ada kalanya diperlukan *custom scalar type* untuk keperluan tertentu. Scalar *type* bukanlah tipe objek, dikarenakan *scalar type* tidak memiliki *fields*.

3. Enums

   Enumeration atau *enums* adalah *scalar type* yang memungkinkan *field* untuk menghasilkan set nilai string yang terbatas.

4. Connections and Lists

   Kemampuan untuk menghubungkan data dari permintaan beberapa jenis data terkait adalah fitur yang sangat penting. Saat membuat suatu daftar tipe objek khusus, GraphQL menyediakan fitur canggih untuk membantu dalam menghubungkan objek satu sama lain, yaitu *One-to-One Connections, One-to-Many Connections, Many-to-Many Connections.*

5. List Of Different Types

   Dalam GraphQL, *list* tidak selalu mengembalikan *type* yang sama. Ada dua cara untuk menangani bidang yang dapat berisi banyak *types* dalam sebuah *schema* di GraphQL yakni dengan *Unions* dan *Interfaces*.

6. Argumets

   Arguments merupakan pasangan *key-value* yang terkait dengan *field* kueri. *Argument* dapat ditambahkan ke dalam setiap *field* pada GraphQL, dan sama seperti *field, Argument* juga memerlukan tipe data, baik dengan *scalar type* maupun *object types* yang tersedia dalam *schema*. *Arguments* memungkinkan pengiriman data yang dapat mempengaruhi hasil dari operasi GraphQL.

7. Mutations

Mutations harus didefinisikan dalam schema. Seperti halnya kueri, *mutations* juga didefinisikan dengan *custom object type*-nya sendiri dan ditambahkan ke dalam schema. Perbedaan dalam mendefinisikan *mutations* dan kueri terletak pada tujuannya, dimana *mutations* perlu dibuat hanya apabila suatu *action* atau *event* dapat merubah sesuatu tentang keadaan aplikasi.

8. Input Types

Input Types merupakan tipe data yang mirip seperti GraphQL *object type*, namun hanya digunakan untuk input *arguments*. *Input types* merupakan kunci untuk mengatur dan menulis GraphQL schema yang jelas. Selain dapat digunakan sebagai *argument* pada *field* manapun, *input type* juga dapat digunakan untuk meningkatkan data paging dan data filtering pada aplikasi. *Input type* meningkatkan dokumentasi schema yang dihasilkan GraphQL dan GraphQL Playground secara otomatis, sehingga membuat API yang digunakan lebih mudah dipahami dan dipelajari, serta akan membuat klien memiliki kekuatan untuk menjalankan kueri yang terorganisir.

2.5 Go

Go adalah sebuah bahasa pemrograman yang digagas oleh Robert Griesemer, Rob Pike, dan Ken Thompson di Google pada tahun 2007. Baru setelah dua tahun, Go kemudian diperkenalkan pada tahun 2009. Go memiliki banyak kemiripan dengan bahasa C seperti aturan penulisan, tipe data dasar, pointer dan kemampuan untuk di-*compile* menjadi bahasa mesin (Donovan & Kemighan, 2016). Saat ini Go atau yang sering disebut dengan Golang (Go Language) terus dikembangkan oleh Google bersama dengan kontributor dari komunitas *open source*.

Go mulai banyak digunakan pada berbagai macam perusahaan baik untuk perusahaan berskala besar maupun stratup yang bergerak dibidang teknologi. Menurut Meyerson (2014) beberapa karakteristik bahasa Go secara garis besar dapat dilihat berdasarkan fundamental, Run-time, dan model package-nya.

1. Fundamental

Go memiliki aturan penulisan yang ringkas dan bersih. Selain itu, bahasa ini juga memiliki desain yang ringan sehingga tidak meningkatkan beban kinerja komputer. Beberapa karakteristik Go seperti interface dapat dipisah dari

implementasinya, tidak ada konversi implisit dan aturan lainnya sehingga dapat mempersingkat serta mempermudah penulisan program.

2. Run-Time

   Go memiliki *garbage collection* yang akan menghapus penggunaan memori yang tidak digunakan pada saat program tersebut dijalankan. Selain itu Go juga mendukung konkurensi untuk meningkatkan kinerja program.

3. Package Model

   Go menggunakan konsep dependensi eksplisit untuk mengelolah dependensi yang digunakan sebuah program. Sehingga waktu yang dibutuhkan ketika melakukan *build* pada program menjadi lebih singkat.

2.6 Redis

Redis, yang dikembangkan oleh Salvatore Sanfilippo pada tahun 2009, merupakan database NoSQL berbasis key-value. Redis menggunakan model client-server dan protokol Redis untuk berkomunikasi. Redis merupakan server single thread yang sangat cepat dan ditulis dalam bahasa C. Nama "Redis" sendiri merupakan singkatan dari "Remote Dictionary Server."

Redis mencapai kinerja tinggi berkat penggunaan penyimpanan data di dalam memori. Redis mendukung lima jenis "value," yaitu strings, lists, sets, hashes, dan sorted sets. Selain itu, Redis juga mendukung penyimpanan data persisten di disk, replikasi untuk meningkatkan kinerja baca, dan sharding untuk meningkatkan kinerja tulis. Sebagai database NoSQL, Redis tidak menggunakan tabel untuk mendefinisikan atau mengatur hubungan antara data.

2.7 Apache JMeter

Apache JMeter adalah program yang digunakan untuk melakukan pengujian kinerja *server*. Teknologi yang diperkenalkan pada tahun 2001 ini merupakan project *open source* dan dapat berjalan diberbagai jenis platform. Apache JMeter menggunakan lisensi Apache yang tidak memiliki pembatasan dalam penggunaan, pendistribusian, ataupun modifikasinya. Fungsi utama dari Apache JMeter untuk menguji performa aplikasi atau layanan dengan mensimulasikan sejumlah besar pengguna atau permintaan. Pada penelitian ini Apache JMeter digunakan untuk mengukur *response time*, *throughput*.

Pengujian beban JMeter memungkinkan pengguna untuk membuat skenario pengujian beban yang mensimulasikan sejumlah besar pengguna untuk menguji seberapa baik aplikasi. Aplikasi Apache JMeter dapat dijalankan secara *Graphical User Interface* (GUI) maupun *Command Line Interface* (CLI). Untuk menampilkan hasil pengujian, JMeter dapat mengubah hasil uji kinerja menjadi file HTML untuk ditampilkan di *browser* (Erinle, 2017).

2.8 *JSON Web Token* (JWT)

JSON Web Token (JWT) adalah sebuah standar terbuka (RFC 7519) yang digunakan untuk mengirimkan informasi yang diverifikasi secara aman antara pihak-pihak dalam bentuk JSON. JWT sering digunakan sebagai metode autentikasi dan pertukaran informasi antara server dan klien dalam aplikasi web dan layanan web. Berikut struktur dari JWT yang terdiri dari tiga bagian yaitu header, payload, dan signature yang dipisahkan oleh tanda titik (“.”) :

- *Header*: Bagian *header* berisi metadata tentang token, seperti jenis token dan algoritma yang digunakan untuk mengenkripsi data. *Header* biasanya berbentuk JSON.
- *Payload*: Bagian payload adalah tempat informasi yang ingin disimpan dalam token ditempatkan. Ini bisa berisi klaim pengguna, data tambahan, atau informasi lain yang diperlukan.
- *Signature*: *Signature* digunakan untuk memverifikasi bahwa token tidak berubah selama perjalanan antara server dan klien. Ini dibuat dengan menggabungkan *header*, *payload*, dan sebuah *secret key* yang hanya diketahui oleh pihak yang menghasilkan token.

JSON Web Token adalah metode yang aman dan efisien untuk mengamankan informasi yang dikirimkan antara pihak-pihak yang berinteraksi dalam aplikasi web dan layanan web. Dengan menggunakan JWT, Anda dapat memastikan bahwa data yang Anda kirim dan terima tetap aman dan tidak berubah selama perjalanan di seluruh jaringan.